ARMY RESEARCH LABORATORY

Scalability vs. Performance

by Daniel M. Pressel

ARL-TR-2596 September 2001

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-TR-2596 **September 2001**

Scalability vs. Performance

Daniel M. Pressel
Computational and Information Sciences Directorate, ARL

Abstract

In the ideal world, the performance of a program running on a supercomputer would always be proportional to the peak speed of the system being used. Furthermore, the program would always achieve a high percentage of peak (e.g., 50% or better). In the real world, this is frequently not the case. Therefore, it is important to distinguish between the following five concepts: (1) performance (run time), (2) ideal speedup, (3) hard scalability (fixed problem size speedup), (4) soft scalability (scaled speedup), and (5) throughput (how long it takes to run a collection of jobs).

This report addresses these concepts and explains their meanings and differences. Hopefully, this will allow readers to evaluate the behavior of programs and computer systems, and most importantly, to evaluate their own expectations for running a program on a particular system or class of systems.

Examples, which demonstrate these concepts, are drawn from a variety of projects and include both problems from multiple computational technology areas (CTAs) and results from outside of the Department of Defense (DOD). In some cases, there will also be theoretical arguments to help better explain the issues.

Acknowledgments

The author would like to thank Steve Schraml and Marek Behr for permission to use their results in this report. He would also like to thank his many colleagues that have graciously assisted him in all aspects of the preparation of this report. He would also like to thank the entire Common High Performance Computing Software Support Initiative (CHSSI) **CFD**-6 team for their assistance in this work as part of that team. Additional acknowledgments go to Tom Kendall, Denice Brown, and the systems staff for all of their help. Finally, he would also like to thank the employees of Business Plus Corp., especially Claudia Coleman and Maria Brady, who assisted in the preparation and editing of this report.

This work was made possible through a grant of computer time by the Department of Defense (DOD) High Performance Computing Modernization (HPCM) Program. Additionally, some of the results mentioned in this work came from projects that were funded as part of the Common High Performance Computing Software Support Initiative (CHSSI) administered by the DOD HPCM Program.

Note: All items in bold are in the Glossary.

INTENTIONALLY LEFT BLANK.

Contents

INTENTIONALLY LEFT BLANK.

List of Figures

INTENTIONALLY LEFT BLANK.

List of Tables

INTENTIONALLY LEFT BLANK.

1. Introduction

In the ideal world, the performance of a program running on a supercomputer would always be proportional to the peak speed of the system being used. Furthermore, the program would always achieve a high percentage of peak (e.g., 50% or better). In the real world, this is frequently not the case. Therefore, it is important to study and discuss performance metrics for parallel systems and programming. Two important uses of these metrics are (1) the evaluation of the behavior of programs and computer systems and (2) the evaluation of expectations for running a program on a particular system or class of systems.

The metrics that will be discussed in this report are (1) performance (run time), (2) ideal speedup, (3) hard scalability (fixed problem size speedup), (4) soft scalability (scaled speedup), and (5) throughput (how long it takes to run a collection of jobs).

The discussion of these metrics will include a mixture of theoretical analysis and experimental results. The experimental results will come from a variety of disciplines but, in all cases, will involve real codes (e.g., no benchmarks) with representative data sets. While the experimental results were obtained using real systems, the use of those systems does not constitute an endorsement of the product. Additionally, just because system A outperforms system B for one data set or program does not imply that that will be the case for all data sets or programs.

2. Performance

Most users are primarily interested in the following issues:

(1) the ability of the computer system to run their job;

(2) the correctness of the results;

(3) how fast does the job run once it starts running;

(4) how long will it take a series of jobs to complete; and

(5) when will the system start running their jobs.

The first, second, and fifth of these issues are beyond the scope of this report. The fourth topic will be discussed in section 6. Performance can be quantified as:

$$\text{Performance} = \begin{matrix}\text{Theoretical Peak}\\ \text{Performance}\end{matrix} \times \begin{matrix}\text{Algorithmic}\\ \text{Efficiency}\end{matrix} \times \begin{matrix}\text{Serial}\\ \text{Efficiency}\end{matrix} \times \begin{matrix}\text{Parallel}\\ \text{Efficiency}\end{matrix}.$$

For many jobs, one can specify either a minimum acceptable level of performance and/or a desirable range for the performance. This need not preclude the achievement of even higher levels of performance. However, there may be resource allocation issues that favor sticking to the desirable range for the performance. What is important to note is that the program with the highest level of parallel efficiency may not be the program with the highest level of algorithmic efficiency* and vice versa. Furthermore, the history of parallel computing contains numerous examples of systems that would scale well, but on which it was notoriously difficult to obtain high levels of serial performance (e.g., the Thinking Machines CM2/CM200, many systems containing the Intel i860 microprocessor, and the Cray T3D [Bailey 1993; Simon and Dagum 1991; Simon et al. 1994; Bailey and Simon 1992; Oberlin 1999]). Therefore, it can be seen that all of the terms in this equation actively contribute to the delivered level of performance. This is a very different point of view from those who stress issues such as the following:

(1) The peak level of performance.

(2) The performance of a machine when running the unlimited size LINPACK benchmark (a benchmark that tends to have a high correlation with the peak speed of a system).

(3) That so long as a system is highly scalable with an efficient interconnect, one can "always" overcome a performance problem by using more processors (Simon et al. 1994).

Instead, what may be needed are combinations of programs and systems to run them on that provide an acceptable range of performance (preferably measured in run time, as opposed to **MFLOPS**) for a reasonable range of problem sizes and/or complexities. For example, if two programs can achieve similar results with similar levels of performance, for an acceptable range of problem sizes, then it is unimportant if the combination of program A and machine A has limited scalability past 64 processors and no scalability past 128 processors, while the combination of program B and machine B has good scalability to hundreds of processors. One might ask, how can this be? Some of the rationale behind this statement are as follows:

(1) If the combination of program B and machine B needs the scalability just to match the performance of program A and machine A then, at best, program B and machine B are equal to program A and machine A.

* Algorithmic efficiency is a concept that can be difficult to measure in an absolute sense. However, it can generally be quantified in a relative sense (e.g., the relative number of floating point operations two programs require to obtain a solution to a particular problem at a specified level of precision), and, in most cases, that is sufficient.

(2) If one needs high levels of scalability to match another system's performance, then the cost effectiveness of the system must be considered.

(3) Scalability well beyond the planned size of a system is of primarily theoretical value.

(4) On most systems, most users have limited allocations and/or limited job priorities. Therefore, the user may find it difficult to use more than a certain number of processors at one time. Again, this results in unlimited levels of scalability being primarily of theoretical value.

Of course, it is also possible that the combination of program B and machine B is not only more scalable, but also performs at least as well as program A and machine A on a per processor basis. In such a case, there may be a strong reason for favoring the combination of program B and machine B.

The following excerpts (Mascagni 1990) should help to demonstrate this point:

> "One of the most intriguing aspects of linear elliptic boundary value problems (BVPs) is their relationship to probability.
>
> .
>
> .
>
> .
>
> It is obvious that this algorithm faithfully implements the collection of statistics implied in equation 2 in an "embarrassingly" parallel fashion. . . . It also makes little difference if we implement this algorithm on a shared or distributed memory machine (or a loosely coupled group of workstations) since there is no interprocessor communication until the statistics are centrally collected.
>
> .
>
> .
>
> .
>
> It is well known that these Monte Carlo methods are much inferior to many deterministic methods for these types of problems."

Additional material on this topic can be found in Singh et al. (1998) and Wang and Tafti (1997).

3. Ideal Speedup

Frequently, it is necessary to predict the performance of a program for a fixed problem size when larger numbers of processors are used. In other cases, one needs to consider the relative merits of running two programs at once (each using half of the processors) vs. running the same programs sequentially (each using all of the processors). Questions such as these lead to the concept of ideal speedup.

The most commonly used definition for ideal speedup is that the speed at which the program runs on a particular machine is proportional to the number of processors being used. Returning to the concepts discussed in section 2, this is equivalent to saying that the parallel efficiency should be 100%.

Clearly from the standpoint of efficiency, unless the parallel efficiency is 100%, it is more efficient to run two programs at once, rather than running them sequentially. However, there are frequently other concerns (e.g., memory requirements and/or minimum performance requirements) that may outweigh this consideration.

There are many reasons why a program will not show linear speedup. Many of these have to do with limitations in the parallelization effort and/or inefficiencies in the hardware. As such, they are considered to be the cause for deviations from ideality. These topics will be readdressed later in this report. However, one can argue that for some algorithms, and in particular, for some approaches to parallelizing those algorithms, that linear speedup is not the ideal speedup. Probably the most common example of this occurs when parallelizing a loop with M iterations when using N processors. If M is within about an order of magnitude of N, then the ideal speedup takes on the appearance of a staircase. This can best be seen in Table 1 and Figure 1.

Table 1. Predicted speedup for a loop with 15 units of parallelism.

Number of Processors	Maximum Units of Parallelism Assigned to a Single Processor	Predicted Speedup
1	15	1.000
2	8	1.875
3	5	3.000
4	4	3.750
5–7	3	5.000
8–14	2	7.500
15	1	15.000

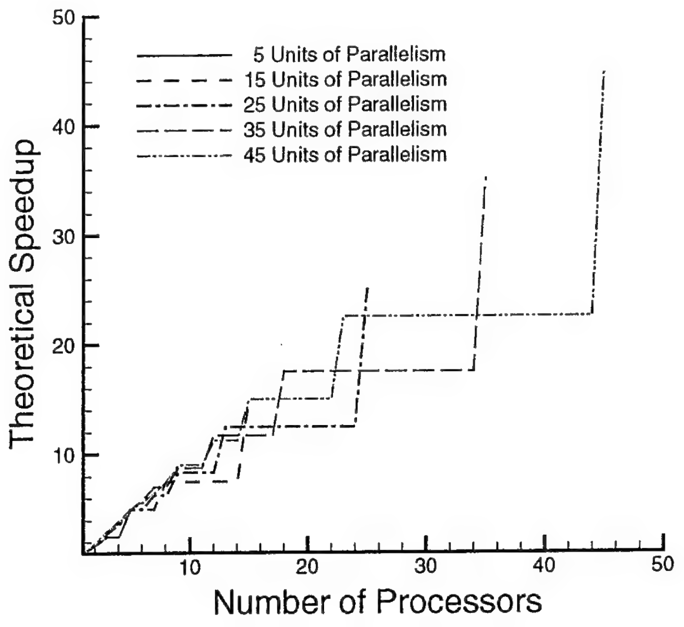


Figure 1. Predicted speedup for a loop with various units of parallelism.

This behavior is commonly seen with programs parallelized using OpenMP and its predecessors (it can also show up in other cases with a limited amount of parallelism [Bettge et al. 1999]). Providing that a program is able to meet its performance criteria, it is probably not appropriate to strongly penalize a program for this type of behavior. Instead, one should take this type of behavior into account when establishing the definition of ideal speedup.

This deviation from linear speedup is not an example of poor load balancing. Poor load balancing occurs when one or more processors receive significantly more work than the remaining processors. In this case, the distribution of work is limited by the limitations of integer arithmetic and, therefore, should be considered to be perfectly balanced (even though some processors might receive one more unit or work than another processor). Similarly, this is not an example of Amdahl's law, since the loop is fully parallelized.

4. Hard Scalability

When discussing the actual scalability of a program, one really needs to talk about the combination of the program, the hardware, and the data set. The earliest metric for scalability is referred to as either hard scalability or fixed size scalability. This assumes that one has a fixed problem to solve and one wants to know how many processors are required to deliver an acceptable level of performance. There can be a number of reasons why the program will fail to deliver ideal speedup. Furthermore, on real distributed memory architectures running real codes and data sets, one frequently finds that large data sets cannot be run using a single processor of an **MPP** (most commonly due to insufficient memory). Smaller data sets that can be run on a single processor of an MPP may have a poor communication-to-computation ratio and, therefore, will show a low level of scalability. As a result of these problems, another metric was proposed, soft scalability, and it will be discussed in section 5.

Many of today's MPPs have powerful enough processors and enough memory per processor to enable many problems to be run on just one or two processors, if only for the purpose of running a scalability study. Therefore, let us briefly consider the three most commonly mentioned reasons for deviations from ideal speedup.

(1) Amdahl's Law: run time = serial run time + parallel run time. As the number of processors approach infinity, the parallel run time will asymptotically approach zero, and the run time will asymptotically approach the serial run time. Therefore, so long as one cannot eliminate the serial run time, there is an upper bound on speed at which a particular machine can run a particular job (see Figure 2).

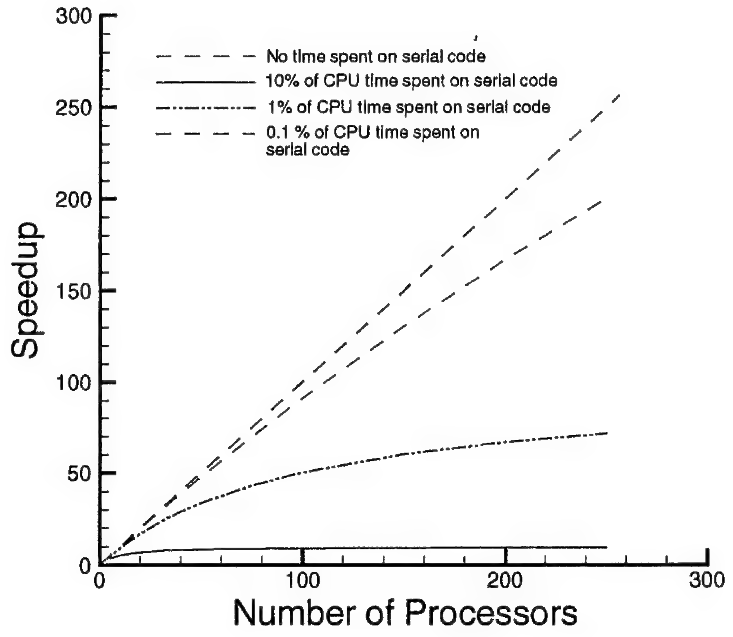
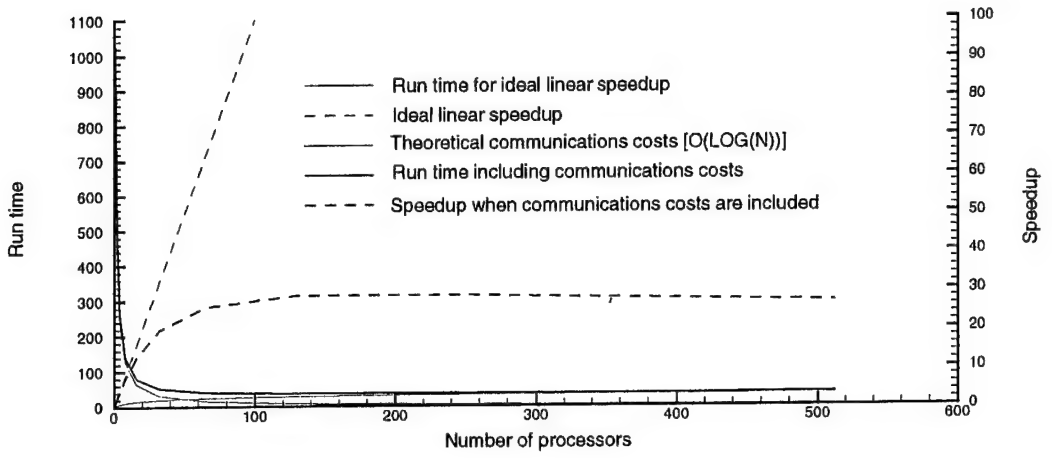


Figure 2. The effect of Amdahl's Law on performance.

(2) Communication costs are nearly always a function of the number of processors being used. In some cases, the function is a weak one (e.g., O(log(N)), while in other cases, it can be much stronger (e.g., O(N)). This now gives us the following: parallel run time = parallel computation time + communication time. Therefore, even if the serial run time is zero, the run time will not asymptotically approach zero. Instead, a plot of the run time as a function of the number of processors used is expected to be U shaped. In other words, there is a small range of processors for which the level of performance will reach a maximum. Past that point, the performance will actually drop off as the number of processors is increased (Almasi and Gottlieb 1994). It is important to note that these costs are primarily a function of three things (the hardware, the number of messages [along with their distribution], and the size of the messages [see Figure 3]).



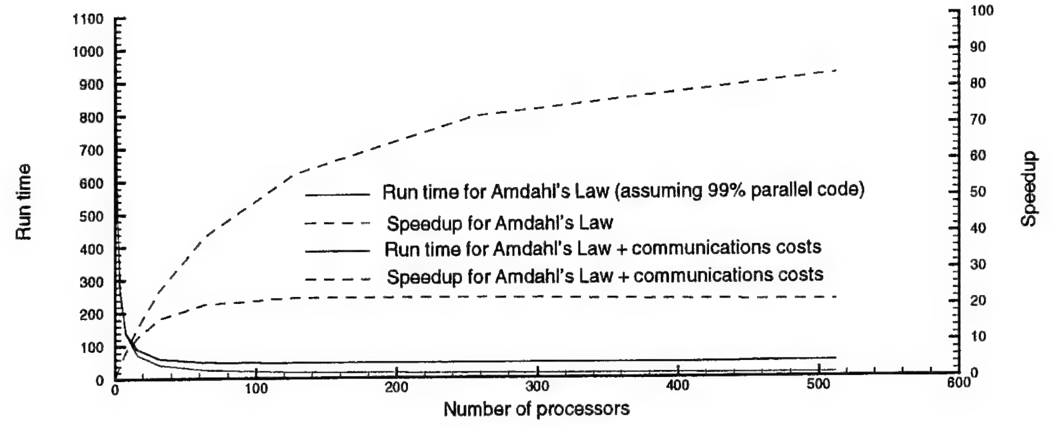


Figure 3. The effect of communications costs on performance.

(3) The load balance: For example, if each part of an airplane's outer surface is assigned to a different processor, then one processor would get most, if not all, of the fuselage. Each wing would be assigned to another processor, and, finally, the tail assembly would be assigned to a small number of processors. Assuming that all of the components are grided at the same resolution, then the processor with the fuselage might be performing upwards of 50% of the work. This would limit the potential for parallel speedup to no more than a factor of 2. Clearly, a better approach is needed. Three commonly used approaches are:

   (a) Domain decomposition, which breaks up the larger zones into more manageable pieces.

   (b) Processing the zones one at a time and parallelizing the processing of the individual zones using loop-level parallelism or other techniques.

   (c) Domain agglomeration, which would assign multiple zones to a single processor. This would be of little value in this case, but might be of value when all of the zones are small, but the range of zone sizes cannot be ignored. Recently, James Taft (a contractor for the NASA Ames Research Laboratory) has been giving talks on some work that he has been doing in this area.

5. Soft Scalability

Soft scalability is also known as scaled speedup and was first proposed by J. L. Gustafson (1988). It proposes that so long as the run time of a job remains roughly constant when the job size and the number of processors increase at proportionally the same rate, then the job should be considered to be scalable. The advantage of this argument is that it allows one to get around the limitations imposed by Amdahl's Law. In fact, for many programs, it can eliminate both that limitation and problems with a poor ratio between communication and computation.

An excellent example of this approach at work was provided by Steve Schraml of the U.S. Army Research Laboratory (ARL), Aberdeen Proving Ground, MD. When running CTH on the SGI R12K Origin 2000 and the SUN HPC 10000s located at the ARL-Major Shared Resource Center (MSRC), he measured the results in Table 2 and Figure 4.

Two important objections to this approach are as follows:

(1) It doesn't address the problem of what to do if the speed at which problem A runs is unacceptable. Presumably, if one runs a problem N times larger using N times as many processors, the speed will still be unacceptable. The obvious answer is to use more processors for the current problem size.

Table 2. The scalability of the SGI R12000 Origin and the SUN HPC 10000 when running CTH.

System	No. of Processors	Measured	Grind Time in microseconds/zone/cycle	
			1 Processor Data[a]	8 Processor Data[a]
SGI Origin	1	36.979	36.979	N/A
	2	20.479	18.490	N/A
	4	10.355	9.2448	N/A
	8	7.2749	4.6224	7.2749
	16	4.0035	2.3112	3.6375
	32	2.0599	1.1556	1.8187
	48	1.4815	0.77040	1.2125
	64	1.2456	0.57780	0.90936
	96	0.73997	0.38520	0.60624
SUN HPC 10000	1	47.558	47.558	N/A
	2	25.622	23.779	N/A
	4	11.875	11.890	N/A
	8	7.0330	5.9448	7.0330
	16	3.7468	2.9724	3.5165
	32	1.8792	1.4862	1.7583
	48	1.2385	0.99079	1.1722
	60	1.1170	0.79263	0.93773
	63	1.1075	0.75489	0.89308
	64	1.1332	0.74309	0.87913

[a] Predictions based on scaling.

However, that raises the question of hard scalability. Potentially, this could result in some problems being run on so many processors that while their overall performance is good, their poor per processor performance might be deemed to be unacceptable. This can be an especially bad problem if it causes one to run out of processors.

(2) This metric cannot be applied to any problem where the parallelism is not directly proportional to the problem size. In particular, when parallelizing the implicit computational fluid dynamics code F3D while using loop-level parallelism, it was discovered that for two important loops, there were dependencies in two out of three directions. Therefore, if each of the dimensions of each zone is doubled, the amount of work increases by a factor of 8, while the parallelism increases by only a factor of 2.

These can be important objections, since using the wrong metric or an inappropriate metric for the case at hand can lead to the wrong conclusions. In some cases, this might result in one choosing a suboptimal solution, while, in other cases, it might result in a project being abandoned entirely.

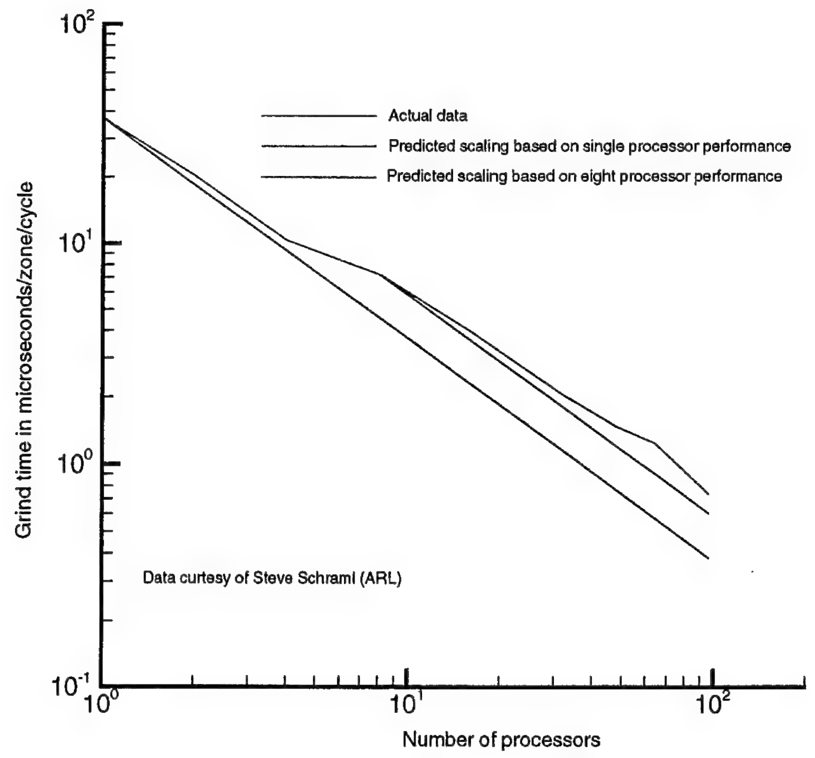


Figure 4. The scalability of the SGI R12000 Origin and the SUN HPC 10000 when running CTH.

6. Throughput

While this metric is important to all users, it can be especially important to those users running parametric studies. These studies can be grouped into three categories:

(1) There are a large number of jobs to run, with no one job requiring a large number of resources. Furthermore, there are no dependencies between the runs, so one can, in theory, run all of them at the same time.

(2) There are a significant number of jobs to run, but they require a moderate-to-large amount of at least one resource (e.g., memory). However, there are few, if any, dependencies between the runs, so one may be able to run a limited number of these jobs at one time.

(3) There are a significant number of jobs to run, with no one job requiring a large number of resources. Unfortunately, there are dependencies between the runs, so one is again limited as to how many jobs can be run at one time.

The importance of these categories is that for a throughput optimized site, the first case might be able to achieve an acceptable level of performance while using a limited number of processors per job. In the other two cases, one will almost always want to use a larger number of processors per job. Therefore, in those cases, the scalability of the job takes on added importance.

An important aspect in terms of throughput is the cost of the hardware in question. While there can be significant variability in the cost of the hardware from one vendor to the next, and from one generation of system to the next within a single vendor's product line, this discussion will ignore those issues. Instead, it will concentrate on the cost of the hardware times the time it is in use for the following three hypothetical system configurations:

(1) Distributed memory MPP with a medium amount of memory (L MBytes), where the cost of the system is 2 * M, where M = the cost of the memory = the cost of everything else.

(2) Distributed memory MPP with a large amount of memory (2 * L MBytes), where the cost of the system is 3 * M, where 2 * M = the cost of the memory, and M = the cost of everything else.

(3) Shared memory MPP with a medium amount of memory per processors (L MBytes), where the cost of the system is 2 * M, where M = the cost of the memory = the cost of everything else.

We will also consider the case of six sets of runs. All of these runs will be assumed to have been parallelized using **MPI** and are assumed to exhibit linear speedup for small numbers of processors. Three of the runs are representative of many CFD applications in that when their work is spread across N processors, the per processor amount of memory required is also decreased by a factor of N. The other three sets of runs are representative of many chemistry applications in that virtually all of the data must be replicated for each processor. Therefore, for this second group of runs, using additional processors will not allow one to run a job that is too big to run on a single processor. The memory requirements for the three jobs from each of the two sets of jobs will be assumed to be L/4, L, and 4L.

Inspection will show that the largest job relying on replication can only be run on the shared memory MPP. Even in this case, it will be "stealing" memory from other jobs' processors. Depending on the workload, this might be acceptable or might require some of the processors to be left unused. Providing that this does not happen often and/or that these jobs represent a small percentage of the total workload, this should be an acceptable solution to the problem of running this type of job. However, if these jobs are more common, then it may be desirable to configure a system specifically to meet the needs of such a job.

Inspection also shows that for the application which does not require the replication of data structures, that for certain problem sizes, one may need to use more than just one processor on a distributed memory system before the job can be run (e.g., the job requiring 4L MBytes of memory requires a minimum of four processors to run on the first of our hypothetical systems). However, most combinations of system type and job size for this class of jobs can be made to work. If one considers the cost of running these jobs, one might assume that the cost would be as follows:

cost for N processors + cost for memory used (e.g., L MBytes).

However, for the distributed memory systems, where the memory is tightly tied to the processors, the actual cost would be as follows:

cost for N processors + cost for the memory associated with

N processors (e.g., N * L MBytes).

From this, one can conclude that regardless of which class of job is run or which size dataset is being run, so long as the job is runable, the cost of running the job on System 1 will always be 2 * M * T1, where T1 is the time to run the job on a single processor (assuming the processor is configured with enough memory to run the job). For System 2, the cost will be 3 * M * T1. Similarly, for System 3, the cost will be 2 * M * T1.

The preceding analysis assumed that a job should only be charged for the resources it is tying up. However, one can also argue that a job should be charged for the resources that it is causing to be tied up. In other words, in order to maintain the ability to run a large memory job on a distributed memory system, the large memory job is causing the system to be configured with extra memory. This has the effect of decreasing the total number of processors that can be purchased and therefore adversely effecting the throughput of jobs that do not require a system with such a generous configuration. There are three main solutions to this problem; which one should be used can be highly site specific as follows:

(1) Arbitrarily limit the amount of memory per processor on a distributed memory MPP, thereby forcing the jobs to live within that limit. In the past, many customers of MPPs had few, if any, choices as to the amount of memory per processor, thereby forcing them into this mode of operation.

(2) Purchase either multiple systems and/or systems composed of nodes with multiple configurations. In this case, one can attempt to more closely match the requirements of the jobs to the available hardware. In general, this solution can be very cost effective and therefore should result in a superior level of throughput.

(3) Purchase at least some shared memory systems to run the jobs requiring the greatest amount of memory per processor. The inherent flexibility of these systems may justify the additional expenses associated with this class

of hardware (something that has been ignored in this discussion up until now). This does not mean that this class of system should be the only class purchased. Nor does it mean that it should represent the majority of the dollars spent. However, it can be an extremely efficient method for supporting a modest number of memory-hungry jobs (frequently referred to as *memory hogs*). Depending on the job mix and the mix of system configurations that were purchased, one can sometimes argue that these systems will pay for themselves by decreasing the amount of memory that the MPP(s) need to be equipped with.

7. Serial Efficiency

Most of this report has dealt with the scalability. Now let us return to the question of serial efficiency. Even if one is running similar programs based on the same algorithm using similar parallelization strategies, differences in serial efficiency can significantly affect the performance of the programs. In particular, we will consider the performance of three versions of the F3D program that was previously mentioned. Marek Behr, formerly of the U.S. Army High Performance Computing Research Center, produced two versions of the code designed to run on distributed memory platforms. One version used SHMEM calls and could be run on either the SGI Origin 2000 or the Cray T3E. The other version of this code used the more portable, but arguably less efficient MPI calls. The third version of the code was written by the author and was based on compiler directives for loop-level parallelism. As such, it could only be run on a shared memory platform and is highly dependent on the design characteristics of the platform being used. Table 3 and Figure 5 contain results from running these codes on several different platforms for a 1-million grid point test case.

From the results in Table 3 and Figure 5, one can see that there are a number of factors which can affect the performance of a program. The peak speed of the processor and the number of processors used are only two of those factors.

Table 3. The performance of various versions of the F3D code when run on modern scalable systems.[a]

System	Peak Processor Speed (MFLOPS)	No. of Processors Used	Version	Speed	
				(time steps/hr)	MFLOPS
SGI R10K O2K	390	8	Compiler Directives	793	1.04E3
SGI R12K O2K	600	8	SHMEM	382	4.99E2
SGI R10K O2K	390	32	Compiler Directives	2138	2.79E3
SGI R12K O2K	600	32	SHMEM	989	1.29E3
	600		Compiler Directives	2877	3.76E3
SGI R10K O2K	390	48	Compiler Directives	2725	3.56E3
SGI R12K O2K	600	48	SHMEM	1083	1.42E3
	600		Compiler Directives	3545	4.63E3
SGI R10K O2K	390	64	Compiler Directives	2601	3.40E3
SGI R12K O2K	600	64	SHMEM	1050	1.37E3
	600		Compiler Directives	3694	4.83E3
SGI R10K O2K	390	88	Compiler Directives	3619	4.73E3
SGI R12K O2K	600	88	SHMEM	1320	1.73E3
	600		Compiler Directives	5087	6.65E3
Cray T3E-1200	1200	8	SHMEM	349	4.56E2
		32		1062	1.39E3
		48		1431	1.87E3
		64		1705	2.23E3
		88		2443	3.19E3
		128		2948	3.85E3
IBM SP 160 (MHz)	640	8	MPI	199	2.60E2
		32		342	4.47E2
		48		420	5.49E2
		64		423	5.52E2
		88		396	5.18E2
Sun HPC 10000	800	8	Compiler Directives	999	1.31E3
		32		2619	3.64E3
		48		3093	4.04E3
		56		3391	4.43E3
		64		2819	3.68E3
HP V-Class	1760	8	Compiler Directives	1632	2.13E3
		14		2392	3.13E3

[a] For additional details, see Behr et al. (2000).

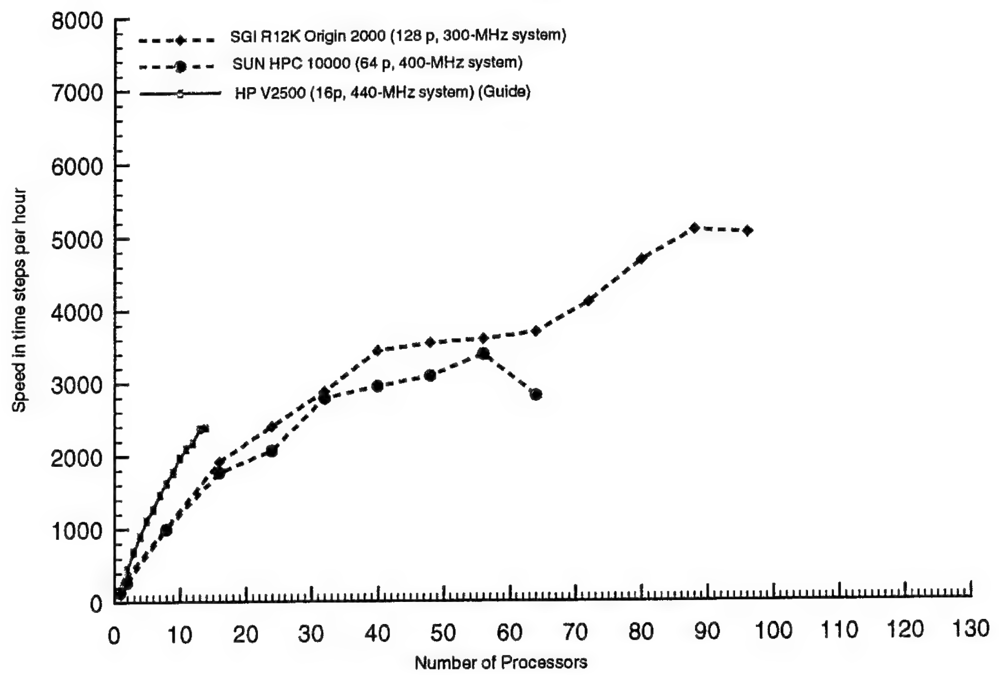


Figure 5(a). The comparative performance of the parallelized RISC optimized version for shared memory platforms of the F3D code.*

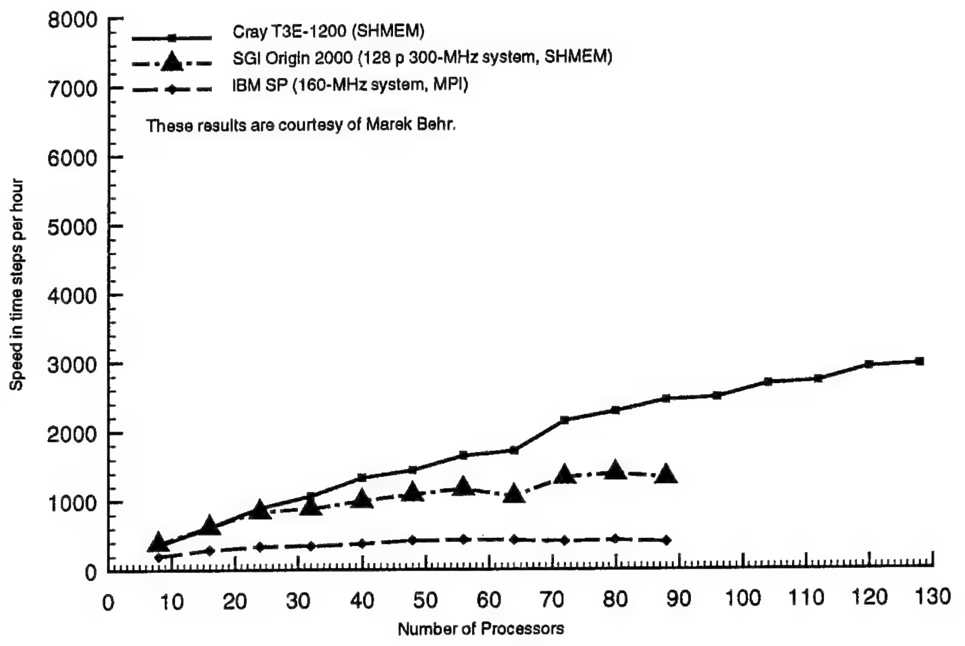


Figure 5(b). The comparative performance of the parallelized RISC optimized version for distributed memory platforms of the F3D code.*

* The speeds have been adjusted to remove startup and termination costs.

8. Conclusions

This report has discussed a number of issues relating to the topics of scalability and performance. It has been shown that for some problems, the ideal speedup will resemble a stair step rather than a straight line. With this concept in hand, two ways for measuring scalability were discussed, with emphasis placed on their strengths and weaknesses. This discussion included examples using these metrics. Hopefully, this report will help the reader in his/her work. In particular, it points out that while scalability is good, most users are concerned with performance and throughput.

9. References

Almasi, G. S., and A. Gottlieb. *Highly Parallel Computing 2nd Edition*. Redwood City, CA: Benjamin/Cummings Publishing Company, 1994.

Bailey, D. H. "RISC Microprocessors and Scientific Computing." Proceedings for Supercomputing 93, 1993.

Bailey, F. R., and H. D. Simon. "Future Directions in Computing and CFD." American Institute of Aeronautics and Astronautics, http://www.nas.nasa.gov/NAS/TechReports/RNRreports/hsimon/RNR-92-019 RNR-92-019.o.html, 1992.

Bettge, T., A. Craig, R. James, W. G. Strand, Jr., and V. Wayland. "Performance of the PCM on the SGI Origin 2000 and the Cray T3E." The 41st Cray User Group Conference, Minneapolis, MN, May 1999.

Gustafson, J. L. "Reevaluating Amdahl's Law." *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, The Association for Computing Machinery, Inc., May 1988.

Mascagni, M. "Parallel Wiener Integral Methods for Elliptic Boundary Value Problems: A Tale of Two Architectures." http://sushi.st.usm.edu /~mascagni/ftp/astfalk.ps, originally published in 1990 in *SIAM News*, vol. 23, no. 4, July 1990.

Oberlin, S. Keynote slides for ISCA'99. The 26th International Symposium on Computer Architecture, http://www.neci.nj.nec.com/isca99/, 1999.

Simon, H. D., and L. Dagum. "Experience in Using SIMD and MIMD Parallelism for Computational Fluid Dynamics." http://www.nas.nasa.gov/NAS /TechReports/RNRreports/ hsimon/RNR-91-014/RNR-91-O14.o.html, 1991.

Simon, H. D., W. R. Van Dalsem, and L. Dagum. "Parallel Computational Fluid Dynamics: Current Status and Future Requirements." http://www.nas.nasa .gov/NAS/TechReports/ RNRreports/hsimon/RNR-92-004/ RNR-92-004. html, 1994.

Singh, K. P., B. Uthup, and L. Ravishanker. "Parallelization of Euler and N-S Code on 32 Node Parallel Super Computer PACE+." Presented at the ADA/DRDO-DERA Workshop on CFD, 1998.

Wang, G., and D. K. Tafti. "Performance Enhancement on Microprocessors With Hierarchical Memory Systems for Solving Large Sparse Linear Systems." *The International Journal of Supercomputing Applications*, February 1997.

INTENTIONALLY LEFT BLANK.

Glossary

AHPCRC	Army High Performance Computing Research Center
CFD	Computational fluid dynamics
MFLOPS	Million floating point operations per second
MIMD	Multiple instruction multiple data
MPI	Message-passing interface
MPP	Massively parallel processor
RISC	Reduced instruction set computer
SIMD	Single instruction multiple data

INTENTIONALLY LEFT BLANK.

NO. OF COPIES	ORGANIZATION
2	DEFENSE TECHNICAL INFORMATION CENTER DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	HQDA DAMO FDT 400 ARMY PENTAGON WASHINGTON DC 20310-0460
1	OSD OUSD(A&T)/ODDR&E(R) DR R J TREW 3800 DEFENSE PENTAGON WASHINGTON DC 20301-3800
1	COMMANDING GENERAL US ARMY MATERIEL CMD AMCRDA TF 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
1	INST FOR ADVNCD TCHNLGY THE UNIV OF TEXAS AT AUSTIN 3925 W BRAKER LN STE 400 AUSTIN TX 78759-5316
1	US MILITARY ACADEMY MATH SCI CTR EXCELLENCE MADN MATH THAYER HALL WEST POINT NY 10996-1786
1	DIRECTOR US ARMY RESEARCH LAB AMSRL D DR D SMITH 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRL CI AI R 2800 POWDER MILL RD ADELPHI MD 20783-1197

NO. OF COPIES	ORGANIZATION
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CI LL 2800 POWDER MILL RD ADELPHI MD 20783-1197
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CI IS T 2800 POWDER MILL RD ADELPHI MD 20783-1197
	ABERDEEN PROVING GROUND
2	DIR USARL AMSRL CI LP (BLDG 305)

NO. OF COPIES	ORGANIZATION
1	PROGRAM DIRECTOR C HENRY 1010 N GLEBE RD STE 510 ARLINGTON VA 22201
1	DPTY PROGRAM DIRECTOR L DAVIS 1010 N. GLEBE RD STE 510 ARLINGTON VA 22201
1	DISTRIBUTED CENTERS PROJECT OFFICER V THOMAS 1010 N GLEBE RD STE 510 ARLINGTON VA 22201
1	HPC CTRS PROJECT MNGR J BAIRD 1010 N GLEBE RD STE 510 ARLINGTON VA 22201
1	CHSSI PROJECT MNGR L PERKINS 1010 N GLEBE RD STE 510 ARLINGTON VA 22201
1	RICE UNIVERSITY MECHANICAL ENGRNG & MATERIALS SCIENCE M BEHR MS 321 6100 MAIN ST HOUSTON TX 77005
1	J OSBURN CODE 5594 4555 OVERLOOK RD BLDG A49 RM 15 WASHINGTON DC 20375-5340
1	NAVAL RSCH LAB J BORIS CODE 6400 4555 OVERLOOK AVE SW WASHINGTON DC 20375-5344
1	WL FIMC B STRANG BLDG 450 2645 FIFTH ST STE 7 WPAFB OH 45433-7913
1	NAVAL RSCH LAB R RAMAMURTI CODE 6410 WASHINGTON DC 20375-5344

NO. OF COPIES	ORGANIZATION
1	ARMY AEROFLIGHT DYNAMICS DIRECTORATE R MEAKIN M S 258 1 MOFFETT FIELD CA 94035-1000
1	NAVAL RSCH LAB HEAD OCEAN DYNAMICS & PREDICTION BRANCH J W MCCAFFREY JR CODE 7320 STENNIS SPACE CENTER MS 39529
1	US AIR FORCE WRIGHT LAB WL FIM J J S SHANG 2645 FIFTH ST STE 6 WPAFB OH 45433-7912
1	US AIR FORCE PHILIPS LAB OLAC PL RKFE CAPT S G WIERSCHKE 10 E SATURN BLVD EDWARDS AFB CA 93524-7680
1	NAVAL RSCH LAB DR D PAPACONSTANTOPOULOS CODE 6390 WASHINGTON DC 20375-5000
1	AIR FORCE RSCH LAB DEHE R PETERKIN 3550 ABERDEEN AVE SE KIRTLAND AFB NM 87117-5776
1	NAVAL RSCH LAB RSCH OCEANOGRAPHER CNMOC G HEBURN BLDG 1020 RM 178 STENNIS SPACE CENTER MS 39529
1	AIR FORCE RSCH LAB INFORMATION DIRECTORATE R W LINDERMAN 26 ELECTRONIC PKWY ROME NY 13441-4514
1	SPAWARSYSCEN D4402 R A WASILAUSKY BLDG 33 RM 0071A 53560 HULL ST SAN DIEGO CA 92152-5001

NO. OF COPIES	ORGANIZATION
1	USAE WATERWAYS EXPERIMENT STATION CEWES HV C J P HOLLAND 3909 HALLS FERRY RD VICKSBURG MS 39180-6199
1	US ARMY CECOM RSCH DEVELOPMENT & ENGRNG CTR AMSEL RD C2 B S PERLMAN FT MONMOUTH NJ 07703
1	SPACE & NAVAL WARFARE SYSTEMS CTR K BROMLEY CODE D7305 BLDG 606 RM 325 53140 SYSTEMS ST SAN DIEGO CA 92152-5001
1	DIRECTOR DEPARTMENT OF ASTRONOMY P WOODWARD 356 PHYSICS BLDG 116 CHURCH ST SE MINNEAPOLIS MN 55455
1	RICE UNIVERSITY MECHANICAL ENGRNG & MATERIALS SCIENCE T TEZDUYAR MS 321 6100 MAIN ST HOUSTON TX 77005
1	ARMY HIGH PERFORMANCE COMPUTING RSCH CTR B BRYAN 1200 WASHINGTON AVE S MINNEAPOLIS MN 55415
1	ARMY HIGH PERFORMANCE COMPUTING RSCH CTR G V CANDLER 1200 WASHINGTON AVE S MINNEAPOLIS MN 55415
1	NAVAL CMD CNTRL & OCEAN SURVEILLANCE CTR L PARNELL NCCOSC RDTE DIV D3603 49590 LASSING RD SAN DIEGO CA 92152-6148

NO. OF COPIES	ORGANIZATION
1	UNIVERSITY OF TENNESSEE COMPUTER SCIENCE DEPT S MOORE 1122 VOLUNTEER BLVD STE 203 KNOXVILLE TN 37996-3450
	ABERDEEN PROVING GROUND
30	DIR USARL AMSRL CI N RADHAKRISHNAN AMSRL CI H C NIETUBICZ AMSRL CI HA D PRESSEL D HISLEY R NAMBURU R VALISETTY D SHIRES R MOHAN M HURLEY P CHUNG J CLARKE C ZOLTANI A MARK AMSRL CI HC D BROWN R PRABHAKARAN T PRESSLEY T KENDALL P MATTHEWS K SMITH AMSRL WT PB J SAHU K HEAVEY P WEINACHT AMSRL WM TC S SCHRAML K KIMSEY S SCHETTLER R COATES AMSRL WM T B BURNS AMSRL WM TA D KLEPONIS M NORMANDIA AMSRL WM BF H EDGE

INTENTIONALLY LEFT BLANK.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 2001	3. REPORT TYPE AND DATES COVERED Final, October 1999–June 2000

4. TITLE AND SUBTITLE Scalability vs. Performance	5. FUNDING NUMBERS 66580373
6. AUTHOR(S) Daniel M. Pressel	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: AMSRL-CI-HC Aberdeen Proving Ground, MD 21005-5067	8. PERFORMING ORGANIZATION REPORT NUMBER ARL-TR-2596
9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)	10.SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT*(Maximum 200 words)*

In the ideal world, the performance of a program running on a supercomputer would always be proportional to the peak speed of the system being used. Furthermore, the program would always achieve a high percentage of peak (e.g., 50% or better). In the real world, this is frequently not the case. Therefore, it is important to distinguish between the following five concepts: (1) performance (run time), (2) ideal speedup, (3) hard scalability (fixed problem size speedup), (4) soft scalability (scaled speedup), and (5) throughput (how long it takes to run a collection of jobs).

This report addresses these concepts and explains their meanings and differences. Hopefully, this will allow readers to evaluate the behavior of programs and computer systems, and most importantly, to evaluate their own expectations for running a program on a particular system or class of systems.

Examples, which demonstrate these concepts, are drawn from a variety of projects and include both problems from multiple computational technology areas (CTAs) and results from outside of the Department of Defense (DOD). In some cases, there will also be theoretical arguments to help better explain the issues.

14. SUBJECT TERMS supercomputer, high performance computing, parallel programming			15. NUMBER OF PAGES 28
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

INTENTIONALLY LEFT BLANK.